REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE -	3. REPORT TYPE AND DATES COVERED FINAL 01 Sep 98 - 28 Feb 99

4. TITLE AND SUBTITLE Rapid Training of GIL Neural Networks	5. FUNDING NUMBERS DAAG55-98-1-0414
6. AUTHOR(S) Peter Kiessler	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Clemson University	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING / MONITORING AGENCY REPORT NUMBER ARO 37543.1-CI-II

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12 a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12 b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Applying generalized inverse learning to a feedforward neural network has been shown to be an effective tool in pattern recognition. The difficult computational step is finding the pseudo-inverse of a matrix. In this paper, we develop an efficient method using differential equations to calculate the pseudo-inverse.

20010416 023

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OR REPORT **UNCLASSIFIED**	18. SECURITY CLASSIFICATION ON THIS PAGE **UNCLASSIFIED**	19. SECURITY CLASSIFICATION OF ABSTRACT **UNCLASSIFIED**	20. LIMITATION OF ABSTRACT **UL**

Rapid Training of GIL Neural Networks

Clark Jeffries[1a] and Peter Kiessler[2b] and Louis Ntasin[b].

[a]Box 12195 IBM Corp., Research Triangle Park, NC 27709
[b]Dept. of Mathematical Sciences, Clemson Univ., Clemson, SC 29632.

Abstract

Applying generalized inverse learning to a feedfoward neural network has been shown to be an effective tool in pattern recognition. The difficult computational step is finding the pseudo-inverse of a matrix. In this paper, we develop an efficient method using differential equations to calculate the pseudo-inverse.

Keywords: Neural networks, generalized inverse learning, pseudo-inverse

1 Introduction

Applying generalized inverse learning (GIL) to a feedfoward neural network has been shown to be an effective tool in pattern recognition. Both Hanson [1] and Kaye [2] were successful in applying GIL to scenarios using real data. The key feature in GIL is that it trains the network in a single iteration. This suggests that there is potential for developing a pattern recognition tool that can be used in the field. However, first we need to overcome the difficult computational step in finding the pseudo-inverse of a matrix. In this paper, we develop efficient methods that calculate the pseudo-inverse, that is, the Moore-Penrose generalized inverse, see [3], via differential equations.

GIL was introduced by Pethel et. al. for for training feedforward neural networks. This is the third project in Clemson university involving GIL. As mentioned above the key step in training the network is in finding the Moore-Penrose inverse of a matrix M. In the first project, Kaye [2] developed a training technique based upon the singular value decomposition of the matrix M. In the second project, Hanson [1] successfully used the procedure developed by Kaye for pattern recognition. The purpose of this project was to develop an algorithm for training the network that was both fast and simple enough to be implemented on a chip. The algorithm we developed satisfies these properties. Not only has the algorithm been successsfully tested but we have provided justification that it converges to the Moore-Penrose inverse of M.

The next step is to apply the algorithm in a tracking problem. While some steps have been made in this direction, more needs to be done. In section (6), we will elaborate on future possibilities in this direction.

The paper is organized as follows. In section 2, we present a brief discussion on the background. Section 3 then discusses the feedfoward neural network and GIL. There we elaborate on the construction of GIL. In section 4 we discuss efficient tecniques on finding the pseudo-inverse. We outline several methods for finding the pseudo-inverse and examine the performance of each method. We conclude the section with a discussion of the guidelines for a stopping rule for the algorithm. In

[1]Clark Jeffries is currently on leave from the Dept. of Mathematical Sciences, Clemson Univ., Clemson, SC 29632.
[2]Correspondence: Email: pkiess@math.clemson.edu, Telephone: 864-656-3281

hand-delivered by M Jackson

section 5 we present convergence results. Lastly, section 6 provides concluding remarks as well as suggestions for further research.

2 Background

Generalized Inverse Learning (GIL) is a technique recently developed for training feedforward, hidden layer neural networks. This technique is based on the Moore-Penrose generalized inverse of a matrix. Since it's introduction by Pethel et al., GIL has proven itself as a fast and efficient tool for pattern recognition. It has been applied in predicting time series data, acoustic data as well as pattern recognition in grayscale images.

GIL as developed by Pethel et al. can be summarized as:

$$G(IA)B = \Theta \tag{1}$$

where I and Θ are input and output matrices respectively, A and B are the weight matrices, G is a nonlinear transfer function, $[G(X)]_{i,j} = \frac{1}{2}[1 + \tanh(X_{i,j})]$ with inverse $[G^{-1}(X)]_{i,j} = \operatorname{arctanh}(2X_{i,j} - 1)$. Training is achieved by an iterative process in which the entries in A and B are adjusted till the quantity

$$E = \sum_n (\Theta_n - \Phi_n)^2$$

is minimized, where Φ is the actual(target) output.

This training technique depends on the left and right Moore-Penrose generalized inverses of $G(IA)$. Let M be an $m \times n$ matrix, then the left and right generalized inverses of M are defined as

$$M_L^{-1} = (M^T M)^{-1} M^T \quad and \quad M_R^{-1} = M^T (M M^T)^{-1}$$

respectively. Let $n \leq m$ and assume M is of rank n, then $M^T M$ has full rank and is thus invertible while MM^T is singular. Thus M_L^{-1} can be computed while M_R^{-1} is undefined. Realizing how restricted GIL was because of such a training technique, a new training approach was proposed (Kaye) using singular value decomposition (SVD), in which the iterative scheme was reduced to the solution of a system of linear equations: $MB = \Theta$ (where $M = G(IA)$). For any $m \times n$ matrix M the singular value decomposition can be written as $M = V\Sigma W^T$, where V and W are orthogonal matrices and Σ is diagonal with the singular values of M. These singular values are the positive square roots of the eigenvalues of $M^T M$. The generalized inverse(pseudo-inverse) of M is therefore $M^\dagger = W\Sigma^{-T}V^T$, where Σ^{-T} is the transpose of Σ with the diagonal entries being the reciprocals of the strictly positive diagonal entries in Σ. For any rectangular system $Ax = b$, SVD gives the best solution in the least squares sense, that is, the solution that minimizes $E = \sum_n (\Theta_n - \Phi_n)^2$. This change in training method led to GIL-SVD. An attractive property of GIL-SVD is it's one step training.

Kaye [2] then applied GIL-SVD to nonlinear systems, chaotic time series data and to periodic functions sampled at regular intervals. Using a map that consisted of a system of nonlinear equations, GIL-SVD in one pass attained the accuracy level that GIL took 15 iterations to attain. This confirmed the superiority of GIL-SVD over GIL.

Using the first 25 points of the chaotic time series map

$$X_{i+1} = 4X_i(1 - X_i), \qquad X_0 \in (0, 1)$$

to train, GIL-SVD was able to predict the 26th with an average accuracy of 10^{-2}. In fact it was shown that the prediction of the next point on the time series map did not significantly depend on the length of the input.

The last application of GIL-SVD by [2] was on periodic functions (sine curve sampled at a constant rate). Direct application of GIL-SVD yielded high errors. A technique called the embedding-dimension was adopted to reduce the error. An optimum embedding dimension of 3 was found and the error in predicting was of the order of 10^{-4}.

Hanson [1] applied GIL-SVD to pattern recognition in grayscale images. To meet up with the requirements of this particular application a version of GIL-SVD was defined as follows: Let an input vector I and a desired output vector D be given. Construct the weight vector A from selected entries in I as

$$a_j = \frac{1}{2 * i_l}, \qquad i_l \in I$$

Define the transfer function $G(x)$ as

$$G(x) = \frac{2e^{kx}}{e^{kx} + e^{k(1-x)}}$$

where k is the gain parameter for $G(x)$. This particular transfer function was chosen for it's algebraic structure (which we shall exploit later). The one setback of this version of GIL-SVD so far has been the rank deficiency of the matrix $G(IA)$. Thus an exact solution for (1) using the SVD technique cannot be guaranteed. To compensate for this effect of rank deficiency, a shift vector C was introduced to absorb the difference between D and the approximate output obtained. Thus for this particular application GIL-SVD was defined as

$$G(IA)B + C = D$$

Using this particular version Hanson showed that GIL-SVD was a perfect classifier of patterns in grayscale images. Errors from patterns that were not of the training type showed a marked difference when compared to errors from patterns that were of the training type. With $G(IA)$ being rank deficient in some cases, the entries of the pseudo-inverse become very large depending on the degree of deficiency. This introduces the problem of "brittle fits", that is, the sensitivity of the system becomes too high thus rendering the network unreliable. Computing reliable least squares solutions to rectangular systems has been a chalenging problem. For notational purposes we shall maintain the name GIL while refering to GIL-SVD.

3 Artificial Neural Networks and GIL

We consider a feedfoward network whose architecture is shown in Figure 1 and use generalized inverse learning (GIL) for training the network. The primary reason for using GIL is that it trains a neural network in a single iteration. This section describes how GIL trains a neural network and

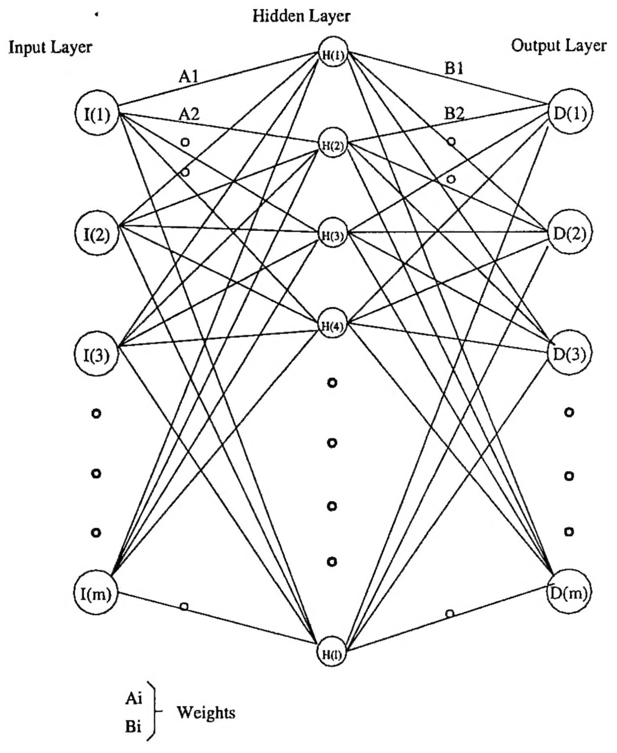

Figure 1: A Standard Feedforward Neural Network.

illustrates that the key computational problem is finding the inverse of a matrix. In the next section, we discuss methods for calculating this inverse.

Given an input vector $I = (I_1, I_2, \ldots, I_m)^T$ and a desired output vector $D = (D_1, D_2, \ldots, D_m)^T$, the neural network represents the function

$$D = G(IA)B + C \tag{2}$$

where A and B are weight vectors, C a shift vector and G is the gain function which are determined as follows.

To determine the weight vector A select n representative input values $(n \leq m)$ among the $\{I_i\}$. Then

$$A = (A_1, A_2, \ldots, A_n) = \left(\frac{1}{2I_1}, \frac{1}{2I_2}, \ldots, \frac{1}{2I_n}\right).$$

Given an $m \times n$ matrix M, $G(M)$ is a $m \times n$ matrix whose $(i,j)^{th}$ element is $g(M_{ij})$ where

$$g(x) = \frac{2e^{kx}}{e^{kx} + e^{k(1-x)}}.$$

Thus, the gain function $G(IA)$ is an $m \times n$ matrix which we assume has rank n.

The weight vector B is found from the equation

$$D = G(IA)B.$$

In the case where $n = m$ then $B = G(IA)^{-1}D$ and we take $C = 0$. When $n < m$, which is usually the case, $D = G(IA)B$ does not have a solution. In this case B is the vector that minimizes the distance between D and $G(IA)B$ and is given by

$$B = G(IA)^{\dagger}D,$$

where $G(IA)^{\dagger}$ is the pseudo-inverse of $G(IA)$. The shift vector C is then found by

$$C = D - G(IA).$$

It follows that the key step in the analysis is in finding $G(IA)^{\dagger}$. We conclude this section by discussing some of the properties of this matrix. To start, consider the function $g(x)$ which is sigmoid and ramps from 0 to 2, taking value 1 when $x = \frac{1}{2}$. The parameter k is called the gain of the sigmoid function and represents the slope of the sigmoid function at $x = \frac{1}{2}$.

For very large values of k, $g(x)$ approximates a step function that maps $x < \frac{1}{2}$ to 0, $x > \frac{1}{2}$ to 2 and $x = \frac{1}{2}$ to 1. So as $k \to \infty$, $G(IA) \to M$, the $m \times n$ matrix

$$M = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 2 & 1 & 0 & \dots & 0 \\ 2 & 2 & 1 & \dots & 0 \\ \dots & & & & \\ 2 & 2 & 2 & \dots & 1 \\ 2 & 2 & 2 & \dots & 2 \\ \dots & & & & \\ 2 & 2 & 2 & \dots & 2 \end{pmatrix}$$

up to permutation of rows.

M has maximal rank and a left pseudo-inverse of M can readily be computed as

$$M^{\dagger} = \begin{pmatrix} 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0 & 0 \dots & 0 \\ 0 & \dots & 0 & -2 & 1 & 0 & \dots & 0 & 0 \dots & 0 \\ 0 & \dots & 0 & 2 & -2 & 1 & \dots & 0 & 0 \dots & 0 \\ 0 & \dots & 0 & \dots & & & & & & \dots \\ 0 & \dots & 0 & \pm 2 & \dots & & \dots & -2 & 1 \dots & 0 \end{pmatrix}$$

$M^{\dagger}M$ is the $n \times n$ identity matrix, I_n, but $MM^{\dagger}$ is not the $m \times m$ identity matrix I_m. Our main task here is to find for any value of k the matrix $M^{\dagger}$, otherwise known as the pseudo-inverse, such that $M^\dagger M$ is I_n and $MM^\dagger$ is as close to I_m as possible. Such an $M^\dagger$ should minimize the sum of squares of the entries of $MM^\dagger - I_m$.

Given any rectangular $m \times n$ matrix M, a singular value decomposition of M can always be written as $M = U\Sigma V^T$, where U ($m \times m$) and V ($n \times n$) are both orthogonal and Σ is an $m \times n$ diagonal matrix with nonnegative diagonal entries (known as the singular values of M) nonincreasing down from the (1,1) position. If M has maximal rank n, then all singular values are positive, otherwise M is rank deficient and some singular values may be zero or close to zero. The pseudo-inverse $M^\dagger = V\Sigma^* U^T$, where Σ^* is the $n \times m$ diagonal matrix with diagonal entries the inverses of the positive diagonal entries in Σ.

Constructing A from the largest n entries of I and letting $k \to \infty$, $G(IA)$ approximates M with pseudo-inverse $M^\dagger$ as below. For this case $M^\dagger M$ is I_n and $||MM^\dagger - I_m||^\ddagger = (m-n)^{\frac{1}{2}}$ which is the theoretical lowest possible.

$$M = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 2 & 1 & 0 & \dots & 0 \\ 2 & 2 & 1 & \dots & 0 \\ \dots & & & & \\ 2 & 2 & 2 & \dots & 1 \\ 2 & 2 & 2 & \dots & 2 \\ \dots & & & & \\ 2 & 2 & 2 & \dots & 2 \end{pmatrix}, \quad M^\dagger = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ -2 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 2 & -2 & 1 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \dots & \\ \pm 2 & \dots & & \dots & -2 & 1 & \dots & 0 \end{pmatrix}$$

4 Efficient Techniques in finding $G(IA)^\dagger$

It is always possible to satisfy (2) with $C \neq 0$. For n=m and sufficiently large k, $G(IA)$ is square and full rank thus invertible giving a solution for (2) with $C = 0$. Our goal therefore at this point is to find the best choices of A and B for any given I and D with the entries in C having the smallest magnitude. This choice of A and B is obtained by finding the pseudo-inverse of $G(IA)$.

Various techniques have been developed for computing this pseudo-inverse, most prominent being the singular value decomposition we mentioned earlier. These techniques are not adequate for our purpose. The ultimate objective of this work is to come up with a scheme that is accurate, fast and stable enough for finite precision arithmetic. The speed required here should be good enough to accommodate real time applications of the GIL. A typical real time application demands that $G(IA)$ be at least of size 150×10.

4.1 The basic idea: ODE approach.

Our technique for finding $G(IA)^\dagger$ is via differential equations. Define a hypersurface of dimension nm and consider that each choice of A and B yields a different position on the hypersurface, corresponding to a particular approximation of $G(IA)^\dagger$. In 2-dimensions a graphical representation of the problem can be summarized as follows. Let $I = (I_1, I_2, \dots, I_m)^T$ be any input vector and let a desired output vector be $D = (D_1, D_2, \dots, D_m)^T$. Then the graph below shows the curve

‡Through out this paper, $||X|| = (\sum_{i,j} |X(i,j)|^2)^{\frac{1}{2}}$.

that represents the possible input/output pairs with various choices of weight vectors A and B. Using the singular value decomposition we can always find that point on the curve that is closest to the given point $(I_1, I_2, \ldots, I_m, D_1, D_2, \ldots, D_m)^T$ with respect to the Euclidean norm. The singular value decomposition inverse that minimizes the distance between the curve and the given point is the pseudo-inverse.

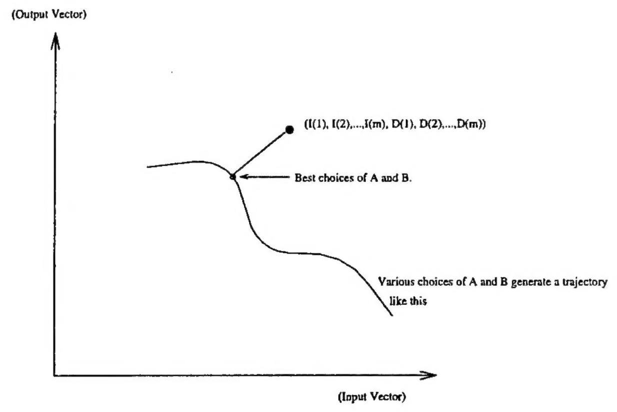
(Output Vector)
(I(1), I(2),...,I(m), D(1), D(2),...,D(m))
Best choices of A and B.
Various choices of A and B generate a trajectory like this
(Input Vector)

Figure 2: The singular value decomposition generates a curve of possible input/output pairs with various choices of A and B.

So given the differential equation

$$\frac{dX}{dt} = f(X)$$

we therefore find an initial point $X(0)$ so that the above equation has a solution $X(t)$ that converges to $G(IA)^\dagger$. We compare two cases, one where f is linear and the other where f is quadratic.

In the linear case we have

$$\frac{dX}{dt} = \{M^T(I_m - MX)\} \tag{3}$$

This is a first order linear homogeneous ordinary differential equation in X. To show that this equation converges to the pseudo-inverse consider the quantity

$$S = \sum_{j=1,i=1} (\delta_{ij} - \sum_{k=1}^{n} M_{ik}X_{kj})^2,$$

where

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

S represents the distance MX is from the identity I_m. The rate of change of S along a trajectory can be shown to be $-\sum_{a,b}(\frac{\partial S}{\partial X_{ab}})^2$. Consequently, S always decreases along all non-constant trajectories. The value of S is finite at any point in the nm-dimensional space. Thus trajectories asymptotically approach the pseudo-inverse. A dynamical system for finding the pseudo-inverse must keep XM close to I_n while changing X so MX approximates I_m.

Various iterative schemes have been defined using this differential equation as the basis for evolution of trajectories on the hypersurface. This differential equation is "well behaved" in that it provides very stable trajectories that guarantee convergence as long as the initial point is on a non-constant trajectory. The main disadvantage of this differential equation and resulting schemes is the convergence rate which is linear in all the cases studied. The rate of change of any trajectory is linear in X which results in a fixed step size Δt.

The schemes developed and studied so far include

$$X(t + \Delta t) = X(t) + \{M^T - X(t)MM^T\}\Delta t \tag{4}$$

which was modified to the dual scheme

$$\begin{aligned} X(t + \Delta t) &= X(t) + \{M^T - M^T M X(t)\}\Delta t \\ X(t + \Delta t) &= X(t) + \{M^T - X(t)MM^T\}\Delta t. \end{aligned} \tag{5}$$

Also schemes in the line of Runge Kutta orders 2 and 4 have been studied. The Newton's method has also been implemented defining the function to be solved as $F(X) = I_m - MX$. This approach showed the expected quadratic convergence rate, but lost some degree of speed due to the lack of a variable step size.

Due to speed limitations in the schemes derived from (3) the following quadratic differential equation was defined.

$$\frac{dX}{dt} = \{X(I_m - MX)\} \tag{6}$$

Since this differential equation defines a continuous map in the hypersurface we can specify a mathematical theory that leads to the difference equation

$$X(k+1) = X(k) + X(k)\{I_m - MX(k)\}\Delta t \tag{7}$$

This scheme has a convergence rate that is at least quadratic thus providing the necessary speed. Due to the selective nature of the scheme all starting points do not converge to the pseudo-inverse. See the appendix for a proof of the optimum choice of Δt and a guarantee of a starting point for this particular class of problems.

4.2 Comparison of the methods

Based on (3) above three different schemes were developed: (4) later modified to (5), a Runge Kutta order 2 scheme and a Runge Kutta order 4 scheme. The Tables(1 and 2) below show typical performance of these schemes on two different matrices. The results are based on 200 iterations for each scheme. Each matrix is a gray scale representation of some picture, some of which are adapted from Hanson [1].

$M1(60 \times 10)$	Error(Left) $I_n - XM$	Error(Right) $I_m - MX$	Largest Entry in $\|X\|$	Number of flops(Millions)
Modified Scheme (5)	2.53	7.63	2.3	17.6
Runge-Kutta Order 2	2.53	7.63	2.3	35.6
Runge-Kutta Order 4	2.53	7.63	2.3	72

Table 1.

To provide a common measuring unit for computational complexity the number of floating point operations (flops) was counted for each scheme. The two examples in Tables 1 and 2 show that the costs to attain the accuracy levels in both tables double as you go from (5) to the Runge Kutta order 2 and then to the Runge Kutta order 4.

$M2(150 \times 10)$	Error(Left) $I_n - XM$	Error(Right) $I_m - MX$	Largest Entry in $\|X\|$	Number of flops(Millions)
Modified Scheme (5)	2.03	12.70	2.2	98.3
Runge-Kutta Order 2	2.03	12.70	2.2	197.3
Runge-Kutta Order 4	2.03	12.70	2.2	396.5

Table 2.

The performance of (7) on four different matrices is shown below. Table (3) is based on 40 iterations while in Table (4) the stopping criteria is based on the size of the entries in X, stopping once the absolute maximum entry exceeds 2 as the iterations progress.

Matrix Size	Error(Left) $I_n - XM$	Error(Right) $I_m - MX$	Largest Entry in $\|X\|$	Number of flops(Millions)
$M(60 \times 10)$	0.00	7.07	3649	1.7
$M(150 \times 10)$	0.00	11.8	91	3.9
$M(175 \times 10)$	0.00	12.8	4.0	4.6
$M(1596 \times 10)$	0.00	40	.15	40.7

Table 3.

¿From Tables 1 to 4 it can be seen that the schemes based on (3) are computationally expensive relative to (7) that is based on (6). Though (7) proves to be computationally less expensive, the schemes based on (3) have an advantage that (7) lacks: global convergence.

Matrix Size	Number of Iterations	Error(Left) $I_n - XM$	Error(Right) $I_m - MX$	Largest Entry in $\|X\|$	Number of flops(Millions)
$M(60 \times 10)$	11	2.00	7.34	2.54	.5
$M(150 \times 10)$	14	1.00	11.9	2.35	1.4
$M(175 \times 10)$	13	0.6	12.9	2.05	1.5
$M(1596 \times 10)$	10	0.0	40	.11	10.4

Table 4.

4.3 Stopping Criteria

In the previous subsection, several examples have shown that the iterative scheme based on (7) converges quickly to the pseudo-inverse. While any stopping criteria should depend on the accuracy of X to the pseudo-inverse, there is another issue we need to consider. This issue is the conditioning of M.

The conditioning of M dramatically affects the magnitude of the elements of X. Recall that the ultimate objective is to implement this scheme with finite bit arithmetic. If we can not guarantee a bound on the magnitude of the entries in $X(k)$, that renders the scheme useless. The table below shows the performance of (7) on four matrices with different conditioning. One can easily recognize the dependence on the conditioning of M of the absolute maximum entry in X.

Matrix size	Largest Entry in $\|X\|$	Condition No. of M
$M(60 \times 10)$	4207	469010
$M(150 \times 10)$	91	23540
$M(175 \times 10)$	4	1229
$M(175 \times 10)$	.15	541

Table 5.

This particular class of problems guarantees a bound on the entries in M, but does not guarantee the conditioning of M. There is one main characteristic of the pseudo-inverse that is of interest to us; it is unique and minimizes $||MX - I_m||$. Consequently, we are guaranteed that (2) is solved as well as possible. With poor conditioning of M, that is, M being close to singular as is the case with some of the M's above, we can not therefore guarantee a bound on the entries in X. The entries in X become very large for M close to singular resulting in brittle fits, rendering the GIL network unreliable. The goal is to choose X so that it avoids brittle fits without compromising the effectiveness of the GIL network.

To develop a stopping rule let $\{X_n\}$ be the sequence of matrices generated by (7) converging to X, where X is the pseudo-inverse of M. Choose N large enough so that for all $n > N$, $||X_n M||$ is within ϵ of 1. Recall that $B = X_n D$ and $C = D - MB$, by an appropriate choice of X_n we can control the magnitude of the entries in B and consequently C. By the choice of the starting point for the scheme, the sequence $\{X_n\}$ starts with terms having known bounds on the magnitude of the entries thereby making it possible for such a choice to exist. Recall also that C, the shift vector, is introduced to take care of the error involved in using the pseudo-inverse. That makes it possible for any left inverse and not necessarily the pseudo-inverse to be used. The closer X_n gets to the pseudo-inverse the better the network is for pattern recognition.

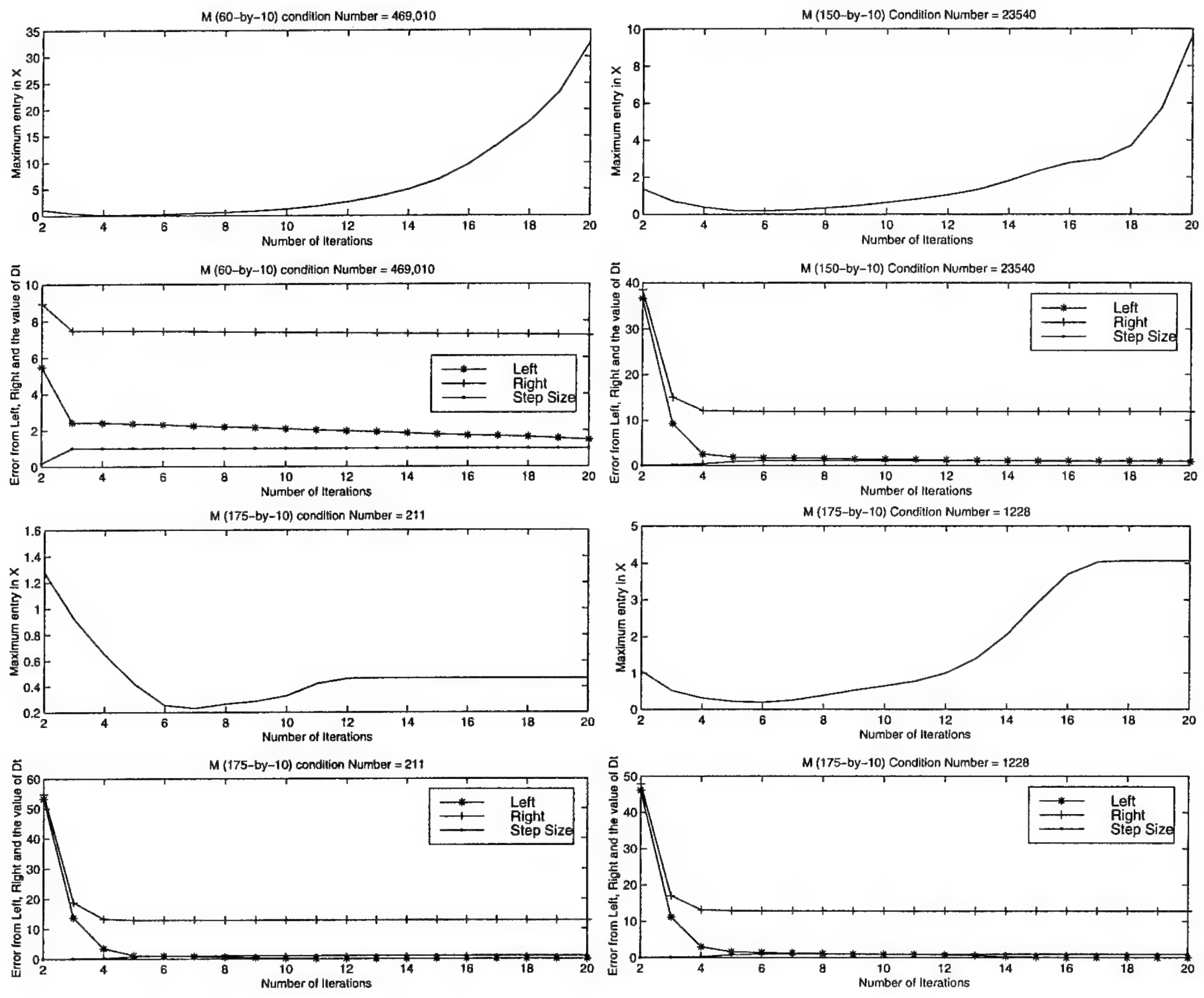

Figure 3: Evolution of four important numerical quantities in the dynamical system.

The graphs above show the evolution of the maximum entry in $|X|$, error levels and step size for four different matrices (Matrix size and condition number given at the top of each graph). These graphs represent part of a careful experimentation with matrices up to size (175×10) in an attempt to determine possible stopping criteria for (7). The graphs in Fig. 3 show that at the fifth iteration the following observations can be made:

1. For each case the largest entry in X has magnitude not exceeding 1.
2. In almost all of the cases the step size is very close to 1.
3. The difference between the error from the right$(||I_m - MX||)$ and the theoretical error from the case with M constructed as $k \to \infty$ does not exceed 1.

4. The error from the left varies from case to case, generally increasing with "poor conditioning" of M.

5 Convergence Results

In this appendix we show that our scheme converges. If m and n are positive integers, $M_{m,n}$ denotes the collection of matrices having m rows and n columns and M_n denotes the collection of square matrices of n rows and columns. For a matrix M, we denote the transpose of M by M^T. The matrix $I_m \in M_m$ denotes the identity matrix.

Let $M \in M_{m,n}$ be real where $m > n$ and assume that M has rank n. From Theorem 7.3.5 in Horn and Johnson [?], M has a singular value decomposition of the form

$$M = V\Sigma W^T$$

where $V \in M_m$ and $W \in M_n$ are unitary and real,

$$\Sigma = \begin{bmatrix} D \\ 0 \end{bmatrix}$$

where $D \in M_n$ is a diagonal matrix whose diagonal elements consist of the strictly positive singular values of M and $0 \in M_{m-n,n}$ is a zero matrix. We note that

$$M^T = W\Sigma^T V^T.$$

The Moore-Penrose inverse for M is the matrix $X \in M_{n,m}$ given by

$$X = W\Sigma^{\dagger} V^T \tag{8}$$

where

$$\Sigma^{\dagger} = \begin{bmatrix} D^{-1} & 0 \end{bmatrix}$$

and $0 \in M_{n,m-n}$ is a zero matrix.

The following procedure is developed for finding the Moore-Penrose inverse of the matrix M. Set $X(0) = M^T$ and for $k = 0, 1, 2, \ldots$

$$X(k+1) = X(k) + X(k)(I_m - MX(k))\Delta t.$$

Lemma 1 *For each $k = 0, 1, 2, \ldots$, the matrix $X(k)$ has a singular value decomposition of the form*

$$X(k) = W\Sigma_k^T V^T.$$

Proof. The proof is by induction. The result is true for $k = 0$ since $X(0) = M^T$ and M^T has the desired decomposition. Now assume the result is true for k. Then

$$\begin{aligned} X(k+1) &= X(k) + X(k)(I_m - MX(k))\Delta t \\ &= W\Sigma_k^T V^T + W\Sigma_k^T V^T (V I_m V^T - V\Sigma W^T W \Sigma_k^T V^T)\Delta t \\ &= W\left[\Sigma_k^T + \Sigma_k^T (I_m - \Sigma\Sigma_k^T)\Delta t\right] V^T. \end{aligned}$$

Thus $X(k+1)$ has the desired decomposition with

$$\Sigma_{k+1}^T = \Sigma_k^T + \Sigma_k^T(I_m - \Sigma\Sigma_k^T)\Delta t$$

which completes the proof.

We write

$$\Sigma_k = \begin{bmatrix} D_k \\ 0 \end{bmatrix}$$

where $D_k \in M_n$ is a diagonal matrix and $0 \in M_{m-n,n}$ is a zero matrix. Moreover,

$$\begin{aligned} MX(k) &= V\Sigma W^T W \Sigma_k^T V^T \\ &= V\Sigma\Sigma_k^T V^T \end{aligned}$$

where $\Sigma\Sigma_k^T \in M_m$ is of the form

$$\Sigma\Sigma_k^T = \begin{bmatrix} DD_k^T & 0 \\ 0 & 0 \end{bmatrix}.$$

Theorem 1 *The Δt's can be chosen so that $X(k)$ converges to the Moore-Penrose inverse X of M.*

Proof. Since the Moore-Penrose inverse is of the form (1), from Lemma 1 it suffices to show that D_k converges to D^{-1}. This is, however, equivalent to showing that DD_k^T converges to I_n.

Let $\lambda_1^k, \lambda_2^k, \ldots, \lambda_n^k$ be the diagonal elements of DD_k^T. Since $D_0 = D$, it follows that for $k = 0$ the λ's are the strictly positive singular values of M. From

$$\begin{aligned} MX(k+1) &= V\Sigma W^T \left\{W \left[\Sigma_k^T + \Sigma_k^T(I_m - \Sigma\Sigma_k^T)\Delta t\right] V^T\right\} \\ &= V\left[\Sigma\Sigma_k^T + \Sigma\Sigma_k^T(I_m - \Sigma\Sigma_k^T)\Delta t\right] V^T \end{aligned}$$

it follows that

$$\lambda_j^{k+1} = \lambda_j^k + \lambda_j^k(1 - \lambda_j^k)\Delta t. \tag{9}$$

For $\Delta t > 0$, the following observations can be made directly from (2):

(i) If $\lambda_j^k = 1$, then $\lambda_j^{k+1} = 1$.

(ii) If $\lambda_j^k < 1$ then $\lambda_j^{k+1} > \lambda_j^k$.

(iii) If $\lambda_j^k > 1$ then $\lambda_j^{k+1} < \lambda_j^k$.

(iv) If we choose $\Delta t = 1/\lambda_j^k$ then

$$\lambda_j^{k+1} = \lambda_j^k + \lambda_j^k(1 - \lambda_j^k)1/\lambda_j^k = 1.$$

Now let $\lambda_{\max}^k = \max\{\lambda_j^k; \lambda_j^k \neq 1\}$ and choose $\Delta t = 1/\lambda_{\max}^k$. Then for every j such that $\lambda_j^k > 1$ we have

$$\begin{aligned} \lambda_j^{k+1} &= \lambda_j^k + (1 - \lambda_j^k)\lambda_j^k/\lambda_{\max}^k \\ &\geq \lambda_j^k + (1 - \lambda_j^k) \\ &= 1. \end{aligned}$$

So that from (iii),

$$\lambda_j^k > \lambda_j^{k+1} \geq 1.$$

For every j such that $\lambda_j^k < 1$ we have

$$\begin{aligned} \lambda_j^{k+1} &= \lambda_j^k + (1 - \lambda_j^k)\lambda_j^k/\lambda_{\max}^k \\ &\leq \lambda_j^k + (1 - \lambda_j^k) \\ &= 1. \end{aligned}$$

So from (ii)

$$\lambda_j^k < \lambda_j^{k+1} \leq 1.$$

From (iv) our choice of Δt guarantees that at each step the largest λ_j^k becomes 1 at the next step. From (i), once a λ_j^k is 1 it stays at 1. Thus choosing Δt in this way the algorithm will converge in n steps to the Moore-Penrose inverse. This completes the proof of the theorem.

Let us make a couple of remarks about the evolution of the Δt's. The sequence of Δt's as given by the proof are not necessarily monotonic. We note that if $\lambda_{\max}^k > 1$ then Δt as chosen in the proof will be less than 1 but greater than the Δt chosen at the previous step. If, however, $\lambda_{\max}^k < 1$, then the Δt will be bigger than 1 but it is not necessarily larger then the Δt at the previous step.

To construct a nondecreasing sequence of Δt's we chose $\Delta t = 1$ whenever $\lambda_{\max}^k < 1$. Note that in this scenario

$$\lambda_j^{k+1} = 1 - (1 - \lambda_j^k)^2. \qquad (10)$$

Corollary 1 *Suppose that for some k_o, $\lambda_{\max}^{k_o} < 1$. Let $\mathcal{J} = \{j \,;\, \lambda_j^{k_o} < 1\}$. Set $\Delta t = 1$ for all $k \geq k_o$. Then for all $j \in \mathcal{J}$.*

$$\lambda_j^k \to 1$$

as $k \to \infty$.

Proof. Define $T : (0,1] \to (0,1]$ by

$$T(z) = 1 - (1 - z)^2.$$

Then

$$1 - T(z) = (1 - z)^2.$$

Pick $z_0 \in (0,1]$ and define for $k = 1, 2, \ldots$

$$z_k = T(z_{k-1}).$$

Then an induction arguement shows that

$$1 - z_k = (1 - z_0)^{2^k}.$$

Thus $z_k \to 1$ as $k \to \infty$.

Choose $j \in \mathcal{J}$ and set $z_0 = \lambda_j^{k_o}$. Then from (10), $z_k = \lambda_j^{k_o+k}$ and the result now follows.

6 Conclusions and Further Research

Hanson showed that GIL is an acceptable neural network for pattern recognition. She used standard SVD tools to find pseudo-inverses. Due to the general nature of the problem at hand and the solution characteristics reqiuired for implementation on a chip, such techniques are not generally acceptable. So a technique for finding the pseudo-inverse of a matrix over which we have control became necessary, which has been the major part of this work. We have in this present work developed such a tool, that incorporates speed and reliability for computing the pseudo-inverse of a matrix, upon which GIL relies for training.

The convergence of the scheme as well as the speed of convergence have both been guaranteed theoretically as well as computationally. The convergence results (section 5) not only guarantee convergence of the numerical scheme but also tell us that if M is of rank n, then in n iterations we converge to the pseudo-inverse using an optimal set of step sizes. Due to the ill-conditioning of M and the short comings of finite bit arithmetic we cannot afford such step sizes. Thus we stick to the largest step sizes that guarantee stability of the scheme while maintaining a high convergence rate.

Also we have studied the evolution of some important quantities in the scheme forming the basis for a stopping criteria while gaining more insight into the evolution of the iterates. This insight enables us to obtain "good" approximations of the pseudo-inverse while avoiding brittle fits. Based on our observations from section (4) a stopping criteria should be application specific. A suitable stopping criteria should take into account speed, accuracy and the environment under which the GIL neural network is implemented. The choice of a stopping criteria should start with speed requirements. The accuracy level needed should then determine if such an environment exists that can provide the necessary accuracy without compromising the speed.

Preliminary results confirmed that the design of a microchip to implement the system of equations in such a network could use as low as 8 bits. Although finite bit arithmetic based on 8 bits slows down the convergence rate of the weights per iteration, the overall computational costs is cheaper.

What we would like to do next is to simulate an entire network for tracking a dynamical system using GIL for vision. To proceed with such simulations knowledge of important parameters in practical scenerios is necessary. For example, the kind of camera available for use in such applications is very important. The camera speed and response time in moving it to new azimuth and elevation angles should form a very critical part of the simulations. Also we need to address the question of the optimal window size. What window size would allow the system to keep tract of the target while keeping the computational costs at a minimum. To summarize, we would like to apply the algorithm to a tracking problem with realistic parameters.

Given the present estimates of speed of training as shown both experimentally and theoretically, this tool promises an acceptable speed in some real time applications. By optimizing the necessary steps that are computationally expensive, we hope to improve the speed of the entire network. The simulations will provide the basis for estimating the overall computational costs of such a network. These estimates will in turn provide the necessary information for a tracking camera experiment in which GIL will provide vision calculations. Finally, we hope to provide enough information for the decision on whether to proceed to hardware or not. We hope to collaborate with hardware experts during this phase of the work as important hardware decisions will have to be made.

Acknowledgments

This work was sponsered by Army Research Office contract DAAG55-98-1-0414.

References

[1] D. A. Hanson. *Structured Neural Networks with Performance Guarantees*, Department of Mathematical Sciences, Clemson University, Clemson SC 29632.

[2] R. L. Kaye. *A New Neural Network Approach to Pattern Classification*, Department of Mathematical Sciences, Clemson University, Clemson SC 29632.

[3] G. H. Golub and C. F. Van Loan. *Matrix Computations*, John Hopkins, 1989.

[4] M. T. Heath. *Scientific Computing: An Introductory Survey*, WCB/McGraw Hill, 1997.